National College of Ireland

Evaluation and Mitigation of Ransomware using Machine Learning

MSc Research Project
MSc Cyber Security

Saurabh Meher
Student ID: x19218044

School of Computing
National College of Ireland

Supervisor: Dr Vanessa Ayala-Rivera

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Saurabh Meher

Student ID: x19218044

Programme: MSc Cyber Security **Year:** 2021

Module: MSc Research Project/Internship

Supervisor: Dr Vanessa Ayala-Rivera

Submission Due Date: 16/12/21

Project Title: Evaluation and Mitigation of Ransomware using Machine Learning

Word Count: 6304 **Page Count** 24

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: SAURABH MEHER

Date: 16/12/21

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Evaluation and Mitigation of Ransomware using Machine Learning

Saurabh Meher
X19218044

Abstract

Ransomware threat has been increasing in the past few years and this pattern is continuing to damage the business and reputation of companies and individuals. As the attackers have an advantage over the cyber security experts, they can easily adapt their attacks to the defences that the experts have implemented. The new advanced malware is then able to go undetected and fail to go under observation. Previous researches mainly focused on malware in general and ways to defend using machine learning and some only focused on the accuracy of their models. The real motivation of this research is to detect malware specifically ransomware which was not achieved in previous research with the focus on increasing the accuracy and precision in the detection. This paper focuses on the detection of ransomware and demonstrates an approach towards detecting portable executable files (PE files) which are categorized as Microsoft's standard malware using machine learning and deep learning. This approach after being tested on a dataset containing 100,000 files that contain malware and benign files using multiple machine learning classifiers such as MLP, Decision tree, and Random Forest will try to reduce the number of false positives and work on getting high accuracy in identifying PE files. The results of this research will help identify and mitigate losses by ransomware attacks on companies and individuals as the machine learning algorithm will detect the ransomware and the user will be notified about the threat. The MLP model outperformed the other models with an accuracy of 99.875% accuracy while also achieving a rate of 0.6% false positive which means that the model can detect malware with high accuracy and precision.

1 Introduction

Ransomware has been the highlight of the malware attacks in recent years where it has been evolved immensely where both the big companies and individuals have been a victim. The cybercriminals target the vulnerability in the victim's security and try to block all the access to the data and services and demand ransom for the return of the data and services[1]. These attacks are successful when a portable executable file (PE) is executed in a system server without being detected by the antivirus as the attackers have evolved their patterns over the years exploiting the signature-based detection where the malware files have multiple polymorphic layers which avoid the general detection of the malware (Crane, 2021).

Ransomware attacks have successfully been executed since the year 1989 when the World Health Organisation's AIDS conference was hit by the attacker who mailed 20,000 floppy disks to the event which contained the ransomware and locked their systems but the IT

[1] https://en.wikipedia.org/wiki/Ransomware

professionals decrypted the malware and most of the devices got decrypted and data was retrieved. Since the inflow of networking from the year 2005, modern variants of ransomware got introduced which included new types of encryptions but the effect of ransomware only increased after the introduction of cryptocurrencies like bitcoin in the year 2010 (Harford, 2021). After that, not only the IT sector but the health, education, finance, banking, and e-commerce sector were majorly hit as the attackers had an easier way of receiving the payment from the victims. After the year 2015, big ransomware such as Petya, Zcryptor, CryptoLocker, WannaCry (Dixon, 2021) was quite evolved from their predecessors in the level of sophistication in attack techniques.

As the attack techniques grow, the defence strategies should be at par with the evolving attacking history. Existing research done in the field of ransomware is mainly done using static and dynamic analysis of the ransomware after it has done damage to the system and then the antivirus can block the attack if the same attack is done in the future. As this is not the case as seen from the history of attacks, the attacks have always been evolved and been a step ahead of the defensive strategies.

> This led us to the following research question: How accurately can machine learning algorithms detect Portable Executable files of ransomware with minimum false positives and false negatives?
>
> To tackle this, the paper suggests the use of machine learning algorithms such as MLP (Multi-layered perceptron), Random Forest, and Decision Tree to detect PE files that have not been identified yet. This selection of models is done based on previous research and their performance. The model which gives the optimum output where the malware is detected with minimum false positives and negatives can be further used to implement ransomware protection for companies.

The dataset on ransomware is not publicly available and due to time constraints, this research is based on the PE files of general malware instead of a ransomware-specific dataset. The creation of a new ransomware dataset would be time-consuming and redundant as it would be a waste of resources and precious time. Therefore, a publicly available dataset of malware that contained the required features to train a machine learning model was chosen and preferred over the other. This research can be taken forward by other researchers where a new dataset that is ransomware-focused can be used to increase the precision of the classifiers used.

The report is structured as follows: In section 2, the related work from other researchers is discussed and why is this research relevant is given. In section 3, the research methodology is summarized and how to methodology was followed in this project is mentioned. In section 4, the design architecture of the framework is elaborated upon. In section 5, the implementation of all the classifiers and how the classification of the models was done. This section also comprised of how the data was clean and transformed for the classification. In section 6, the results of all the classifiers were discussed and the output of the confusion matrix is discussed. In section 7, the conclusion of the research is stated and how it affects future work and how it can be taken further.

2 Related Work

There have been many recent changes in the way that ransomware attack has been discussed above on how it was evolved till the current date. The emerging ransomware threats have become overwhelming and many researchers have faced the same issue on how to reduce the impact of the malware attack. These researchers have focused on main malware in general but as we have seen how the attacks have been growing due to the pandemic and the use of ransomware-as-a-service how easy it is to target companies and individuals. In this section, we are going to discuss the challenges and hardships that researchers faced while detecting malware and how they could have overcome these challenges.

2.1 Emerging threat

According to (Sherif Saad, 2019) the authors, felt that the commercialization of malware increased the challenges faced by the malware analysts, and defence against it felt impossible. This increase will force the malware analysts to think of creative strategies to act against it and think up new detection strategies. The programming libraries have evolved so much that feature-rich computing paradigms give malware an advantage and the defence of the new generation against the new generation of malware is not as effective as it should have. The authors also suggested some mitigation solutions such as disposable micro detectors and analyst friendly interpretation where both these techniques involved machine learning detection models would be inserted at cheap, small instances which are micro-detectors, and interpretation of those detectors would identify potential malware over the generalization of them or even overfitting of the results that could give the results in false positives and false negatives.

2.2 Taxonomy for malware detection using machine learning

(Asaf Shabtai, 2012) along with other researchers presented the taxonomy for malware detection using deep learning and machine learning algorithms using some feature selection (based on Artificial Neural Networks) and feature types technique in their literature. They developed a framework that detected android malware which was a Host-based Malware Detection system that would be employed to detect continuous features and events occurring on devices and the ML detectors would classify the files as benign and malware files respectively.

2.3 Detection Methods and Techniques

There are in general three techniques for the detection of malware: 1) Static analysis and Dynamic analysis 2) Signature-based detection 3) heuristic and behaviour-based detection

methods. Most of the detection techniques are based on dynamic analysis of the malicious software as stated by (Z. Bazrafshan, 2016) and co-authors of the research that also discussed how the malware keep themselves concealed and stay within the system without revealing their information. Unfortunately, they do not discuss the detection techniques in detail mainly the dynamic and static analysis. The above mentioned is also discussed in the paper by (Alireza Souri, 2018) in their research. Adding to that, they also discussed the challenges related to the malware detection approach and features and factors that affect the classification of malware in data mining. They also gave information on the structure of significant methods in malware detection.

2.4 Portable Executable Files

In the research conducted by (Daniele Ucci, 2018) and his colleagues, they categorized what their main task was, features that could be extracted from malware i.e., the Portable Executable files (PE files), and what all machine learning algorithms could be used to accurately detect PE files. They stuck to the traditional methods of machine learning and did not look into advanced machine learning technologies such as deep learning and different multimodal methods. Whereas (Y. Ye, 2018) covered feature selection (Fang Zhiyang, 2019) and extraction which is the traditional method used to detect malware and PE files. However, the dynamic analysis and dynamic feature selection such as the opcodes, API traces were missing. The multimodal approach was also missing along with deep learning methods to detect malware which could be useful for further research was missing.

Even though the previous papers were insightful they only focused on malware as a whole. This paper focuses on malware but specifically, ransomware, and this framework can work against ransomware and other malware with maximum accuracy and low level of losses in false positives and false negatives.

3 Research Methodology

In this section, the methodology of how the data was collected and processed for the application of machine learning algorithms. The classifiers used for the classification using the features in the dataset to accurately predict the malware are also discussed. For the research methodology, using the CRISP-DM methodology the research took guidance and followed its steps. Following are the steps for CRISP-DM methodology and how it was helpful in this research.

3.1 Methodology

CRISP-DM (IBM, 2021) is a robust and well-defined methodology for data mining and machine learning projects. It provides a structure that is practical, flexible, and useful when it comes to business issues involving machine learning problems. In this research, it was considered to go with this methodology as guidance as it provided the means and support that the project needed.

The following is the working CRISP-DM methodology. First, the business understanding of the research is taken into consideration, then how the data was collected and transformed for the samples to be prepared. Using the classifiers, models are created after this and then evaluation is done where the measurements and calculations are performed on the raw data for further evaluation of the data. If there are changes in this section then we go back to the understanding of the project and evaluate our requirements accordingly and the data is again processed into cleaning and transforming. Once the above steps are done, the deployment of the framework is done on the dataset where the statistical techniques used on the data are inferred and conclusions are drawn out from the classification.

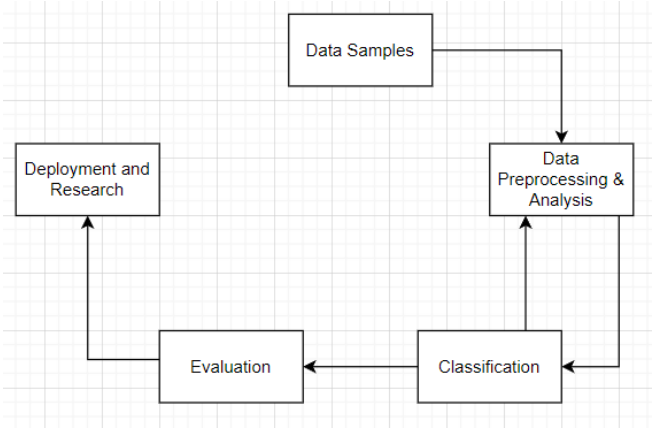


Figure 1: Malware detection flowchart

The above flowchart is customized based on CRISP-DM for the detection of ransomware in this research paper. The detection of ransomware is done using machine learning algorithms such as MLP, Decision Tree, and Random Forest. Initially, the problem with ransomware is understood and why is this research important is discussed. Then the data samples are collected containing malware and benign files and looking at the problem which is ransomware, in general, is on the rise. After the data is collected, data processing is done where the data is transformed, on which machine learning models are applied to classify the files as malware or benign. Once the models classify the dataset, the evaluation of the models is necessary to know which model will be optimum in the real-world scenario. Based on this evaluation, this research can be taken forward to the deployment stage where this can be used against new ransomware that is unknown to the users. Researchers can take this research as a base and work on the detection of ransomware with a better and larger dataset to improve the accuracy of models.

3.2 Language and Machine Environment

Python is an object-oriented, high-level programming language with very rich machine learning-specific libraries and developing frameworks. For cleaning and preparing the data, libraries such as NumPy, scipy, pandas are used which makes the task much easier. The simplicity of the coding language is also remarkable where advanced machine learning concepts can also be used with ease. Python also has a huge community as a result development and addition of new features becomes easy. Python is the most sensible choice for a machine learning development project along with using data from Kaggle where re-usability of code associated with iterative development along with other researchers is possible (Kaggle, 2021). The equipment used in the development of the framework was relatively powerful to produce effective machine learning and deep learning models in the given time frame: 16 GB of RAM with Intel 6 core processors and 12 logical processors (Intel, 2012). Specifications are given as follows:

Processor - Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 2592 Mhz, 6 Core(s), 12 Logical Processor(s) (Lenovo, 2019).

3.3 Dataset

The malware and benign files are equally distributed in the dataset with 50% of malware files and 50% of benign files. This malware dataset has features and columns that can be used for prediction and the importance of each feature is different from each other. Therefore, some of the values need to be processed out while doing the classification.

Initially, an imbalanced dataset was taken into consideration where malware files were dominated by 70% of the whole dataset which provided imbalanced results upon which the dataset was updated and a balanced dataset was taken where malware and benign files had equal distribution in the dataset.

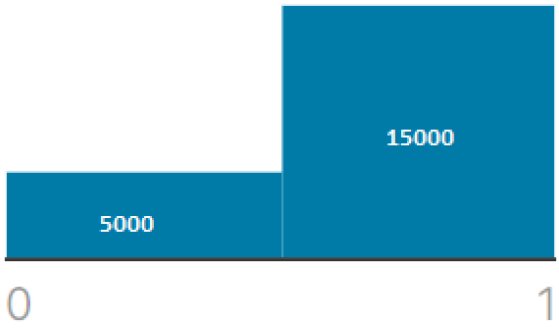


Figure 2: Class Distribution in the unbalanced dataset

The above figure depicts the class distribution of malware in the unbalanced dataset where 0 shows the benign files and 1 shows the malware files in the dataset. As this would not give us the optimal result that is expected, it was not selected and a new balanced dataset was chosen instead of this dataset.

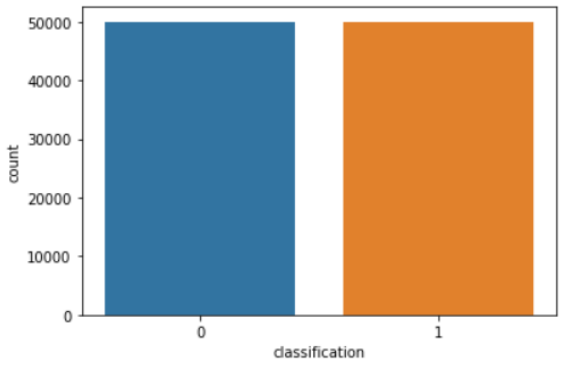
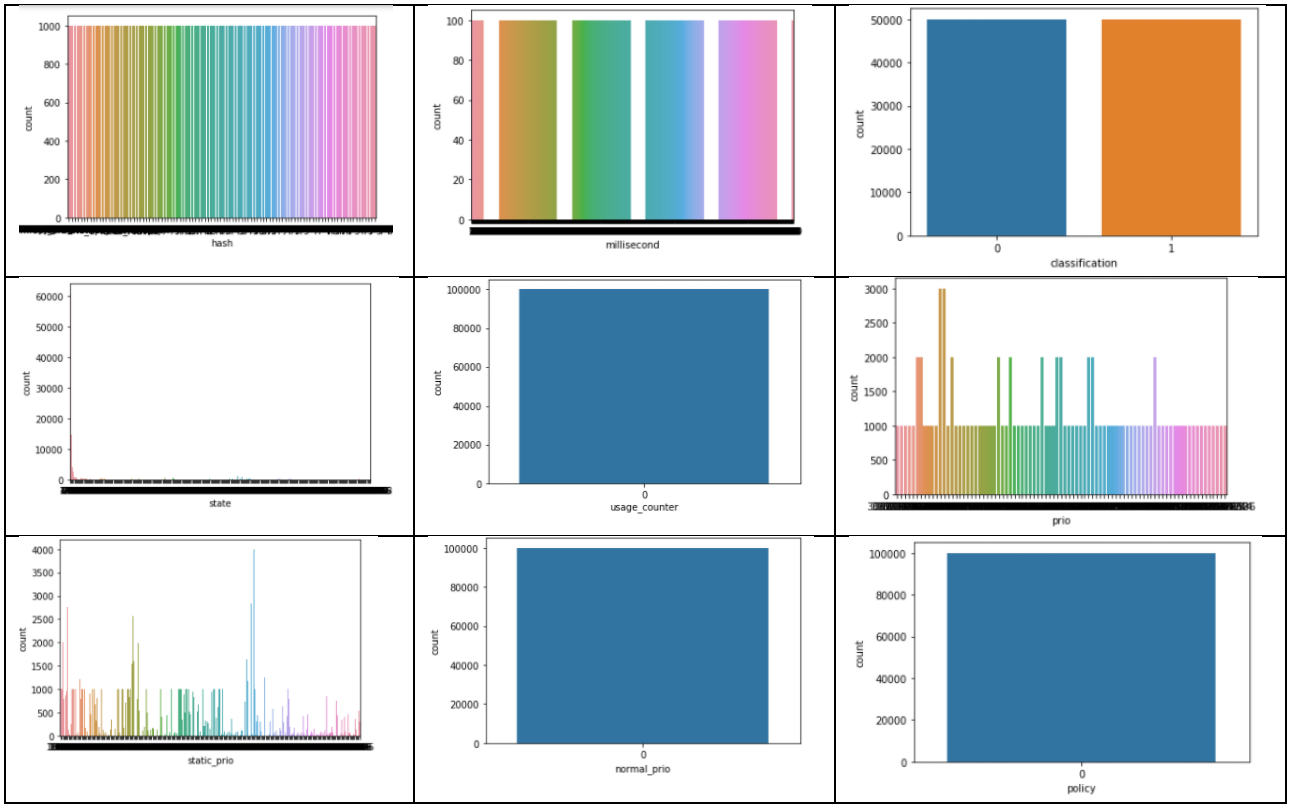


Figure 3: Class Distribution in the balanced dataset

Table 1 shows the feature distribution of the entire dataset. This helps us in visualizing the dataset and knowing which features are necessary for the classification during the modelling phase. If an incorrect feature is applied to the classifiers, then the model will train incorrectly and the results will also come out to be incorrect. As this research is focused on precision and accuracy, the visualization of the dataset is necessary to validate the features before the classification. The following table shows the distribution of the dataset in bar graphs and histograms.



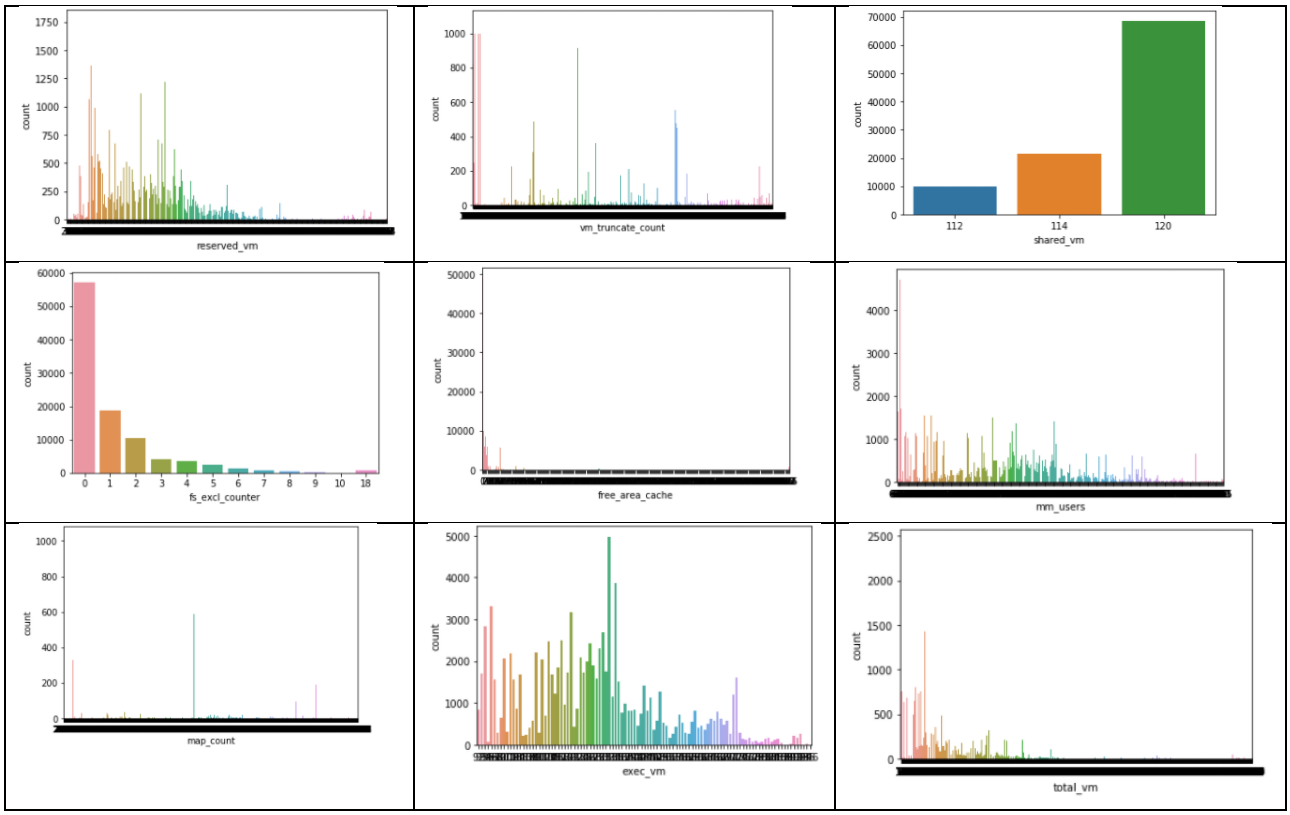


Table 1: Feature Distribution

4 Design Specification

Following is the machine learning architecture followed while developing this approach:
The dataset is first collected and passed to the pre-processing phase. In this phase, data processing, data splitting, and data transformation are done. In data processing, the unused classifiers are removed from the classification. The data is divided into 2 parts: training and testing. 80% of the data is allotted to the train section whereas the other 20% is for the testing section. On this whole data using ANOVA, feature selection is done to sort out the features which have the most significance in the classification. Once the data preprocessing is done, this data is trained and tested using the classifiers and the results are obtained. The results obtained show us the detection of malware is done successfully or not and with what efficiency and precision. The comparison between the models' performance is evaluated and the model which performed the best is selected as the prime model which is optimum for the detection of ransomware using machine learning.

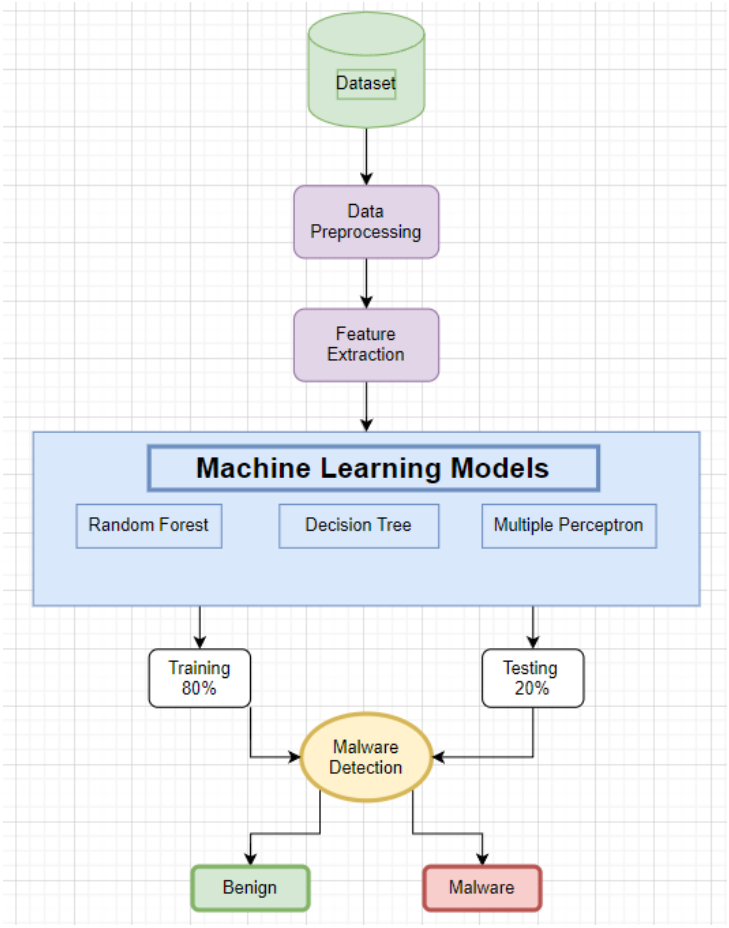


Figure 4: System Architecture

The algorithms chosen in this section are inferred from the related works from the other researches. They are chosen based on their performance recorded in their works and their result. They performed pretty well and have good results in the works of other researchers and were chosen for the same.

From the above classifiers, the outputs produced from them will be discussed in detail in the next section. Once the classification is done using the above classifiers, the next step is to use the feature extracted dataset where the results of the above classifiers are recorded. The accuracy and the false positive count are calculated and tried to make as accurate as possible for the above dataset. Once we have the preferred results, we can infer conclusions based on the result of the above discussion.

5 Implementation

In this section, the implementation of the artefact will be discussed and the implementation of tools and methods will also be discussed. The classifiers will be discussed in detail and the evaluation of the output given by them will be discussed in the next section. The design architecture of this research indicates that the data is divided for training and testing purposes in the proportion of 80-20 for training and testing respectively.

5.1 Libraries

This research uses the programming language Python to conduct all its machine learning conducts using the help of libraries. The libraries used will be discussed below:

As mentioned before, Python is a very flexible language and has a big community where developers can help each other to create libraries of their own to minimize the time of coding. Machine Learning can be done easily with the help of the libraries which are created for this purpose: TensorFlow, NumPy, Pandas, Scikit-learn, Seaborn, Matplotlib. The mentioned libraries are used in the artefact that was developed in this research to detect PE files and classify them as malware or benign files according to the machine learning classifiers passed through them.

TensorFlow: It has excellent services and functionality in terms of deep learning and machine learning capabilities. (Shetty, 2018) In this, we can define our functions and build our models as we see fit for the question and user requirements. Building and training models using this library is very easy using the high-level Keras API which is used to develop the multiple perceptron model in this research.

NumPy: This library is used for the mathematical calculations that are associated with machine learning and the operations that can be done on arrays (Numpy, 2010). It is a powerful tool that adds effectiveness when dealing with data structures in Python helps in matrices calculation and array calculations. In this project, it is used to find the false-positive values from the confusion matrix used in the Random Forest Classifier.

Scikit-Learn: This library is used to call the classifiers in this approach to classify the malware and benign from the dataset. It is also used for feature extraction and data pre-processing.

Seaborn & Matplotlib: These libraries are used for data visualization. They are used to plot graphs and visualize the dataset better for evaluation of dataset and after plotting of the output, it helps in the evaluation of the outputs as well.

Jupyter Notebook: This is the web application that is used to develop this artefact. In this environment, code blocks can be created and executed separately in the Python programming language. Data cleaning and transformation, machine learning algorithms, mathematical calculation are all possible using this web application. The tasks are easier than in any other environment. For developing machine learning projects, Jupyter Notebook is always preferred.

5.2 Data Processing

Once the data visualization is done, the features that are not required can be removed. The features such as 'hash', 'classification', 'vm_truncate_count', 'shared_vm', 'exec_vm', 'nvcsw', 'maj_flt' and 'utime' are features that cannot be used for the classification of the dataset and must be removed before running the model classifiers on the dataset. As these features have values such as constants, binary values, hashes, and bit values they cannot be included as features for the classification as they would not add any value to the prediction. The data must be cleaned and transformed, where if there are empty values must be removed or replaced by another value. After doing the above steps there is another step that is considered where if the data is unbalanced, for better results it is advised that using the methods such as over-sampling and under-sampling are used, where data is reduced or increased whichever the case of the dataset and then sufficient data is kept in the dataset which will provide proper results (Ye Wu, 2012).
The code to drop and clean the dataset from the actual features which are not useful for the training of the model is given as follows:

```
In [12]: #Data Processing
         X = data.drop(["hash","classification",'vm_truncate_count','shared_vm','exec_vm','nvcsw','maj_flt','utime'],axis=1)
         Y = data["classification"]
```

Figure 5: Removal of unimportant features

As for the data splitting for testing and training the data, it is done in an 80%-20% where the majority of the data is split randomly for training the data and the rest is untouched for the testing of the results from the training using the respective classifiers. The splitting of data is done in a random fashion where the random_state = 1 depicts that the random state is set to true and the values taken are in random order. This processing of the dataset is done in the below code snippet:

```
In [58]: #Dataset spliting for testing and training
         from sklearn.model_selection import train_test_split
         x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.2,random_state=1)

         from sklearn.preprocessing import StandardScaler

         scaler = StandardScaler()
```

Figure 6: Dataset Splitting

Here the StandardScaler function will normalize the values of the features such that the distribution will have a mean value of 0 and standard deviation of 1 before applying any machine learning algorithm.

```
In [35]: #Data Transformation
         x_train = scaler.fit_transform(x_train)
         x_test = scaler.fit_transform(x_test)
```

Figure 7: Data Transformation

Figure 7 depicts the transformation of the data where data is transformed to scalar and unimportant features are removed from the dataset for the classifiers to classify with higher

accuracy. There were in total 35 columns in the dataset which is used as features but 8 of the columns did not meet the requirements for the classification of malware.

```
#Feature Selection
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression

anova = SelectKBest(score_func=f_regression, k=6)
x_train_imp = anova.fit_transform(x_train,y_train)
x_test_imp = anova.transform(x_test)


anova.get_support()
```

```
array([False, False, False, False,  True, False, False, False, False,
       False, False, False,  True, False, False,  True, False,  True,
       False,  True, False,  True, False, False, False, False, False])
```

```
X.columns
```

```
Index(['millisecond', 'state', 'usage_counter', 'prio', 'static_prio',
       'normal_prio', 'policy', 'vm_pgoff', 'task_size', 'cached_hole_size',
       'free_area_cache', 'mm_users', 'map_count', 'hiwater_rss', 'total_vm',
       'reserved_vm', 'nr_ptes', 'end_data', 'last_interval', 'nivcsw',
       'min_flt', 'fs_excl_counter', 'lock', 'stime', 'gtime', 'cgtime',
       'signal_nvcsw'],
      dtype='object')
```

Figure 8: Feature Selection

The feature selection and extraction, model classification selection, and data pre-processing are all done in this section. The data is transformed and modified in a scalar fit so that calculation of the mean and the variance of each of the features that is present in the data can be done. For selecting the best features from the dataset, ANOVA was used. Using ANOVA, the proportion of variance is calculated for the features to the total features in the dataset. This means that features with a high proportion are considered in the feature selection in the dataset. Figure 8 shows values generated after feature selection in which the true values can be assumed as fit for use in our prediction model to obtain higher accuracy.

5.3 Classification Model Implementation

5.3.1 Multiple Perceptron

The prediction level of a multiple perceptron model works based on the levels or the hierarchy of the networks. It is a neural network where there are multiple layers which consist of an input layer, a hidden layer, and an output layer.

Mutiple Perceptron

In [16]:
```
input_size = 27

#Number of Outputs
output_size = 2

# Use same hidden layer size for both hidden layers. Not a necessity.
hidden_layer_size = 50

# define how the model will look like
model = tf.keras.Sequential([
    tf.keras.layers.Dense(hidden_layer_size, input_shape=(input_size,), activation='relu'), # 1st hidden layer
    tf.keras.layers.Dense(hidden_layer_size, activation='relu'),
    tf.keras.layers.Dense(hidden_layer_size, activation='relu'),
    tf.keras.layers.Dense(hidden_layer_size, activation='relu'),
    tf.keras.layers.Dense(hidden_layer_size, activation='relu'),
    tf.keras.layers.Dense(hidden_layer_size, activation='relu'),
    tf.keras.layers.Dense(output_size, activation='softmax') # output layer
])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 50)                1400

 dense_1 (Dense)             (None, 50)                2550

 dense_2 (Dense)             (None, 50)                2550

 dense_3 (Dense)             (None, 50)                2550

 dense_4 (Dense)             (None, 50)                2550

 dense_5 (Dense)             (None, 50)                2550

 dense_6 (Dense)             (None, 2)                 102

=================================================================
Total params: 14,252
Trainable params: 14,252
Non-trainable params: 0
_________________________________________________________________
```

Figure 9: MLP model implementation

In our research, we have initialized the size of the hidden layer as 50 and used 6 hidden layers. These values are determined using the trial and error method where optimization of the model was kept in mind and with more depth in the hidden layers, the optimum results were obtained. The hyperparameters used are activation and the size of the layers are specified for each of the layers be it the input layer, hidden layer, or the output layer. Keras deep learning module is used from the TensorFlow library where the input layer and the hidden layer have used the 'relu' activation hyperparameter and the output layer has used the 'softmax' activation parameter.

To further optimize this model the model.compile optimizer has been used with the optimizer adam and reduce the losses used is the 'sparse_categorical_crossentropy' is used to improve the accuracy. Optimization of the model increases the chances of higher accuracy and the cost of the model can also be controlled.

In [17]:
```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Figure 10: Optimizing the MLP model

In this model, an epoch of 20 is taken where the training dataset will pass through the dataset 20 times. An epoch is the number of iterations that the training dataset will pass through the model to increase the accuracy of the model. A maximum number of epochs has been chosen as this will give a more precise and accurate result. This is useful when the dataset is especially very large, creating more than 1 epoch will help the dataset to run in batches and increase the efficiency of the model.

```
In [18]: # set the batch size
         batch_size = 100

         # set a maximum number of training epochs
         max_epochs = 20

         early_stopping = tf.keras.callbacks.EarlyStopping(patience=2)

In [19]: result = model.fit(x=x_train,
                            y=y_train,
                            batch_size=batch_size,
                            epochs=max_epochs,
                            verbose=1,
                            #callbacks=[early_stopping],
                            validation_split=0.2)

Epoch 1/20
640/640 [==============================] - 1s 2ms/step - loss: 0.0723 - accuracy: 0.9753 - val_loss: 0.0117 - val_accuracy:
0.9965
Epoch 2/20
640/640 [==============================] - 1s 1ms/step - loss: 0.0101 - accuracy: 0.9973 - val_loss: 0.0027 - val_accuracy:
0.9992
Epoch 3/20
640/640 [==============================] - 1s 1ms/step - loss: 0.0033 - accuracy: 0.9990 - val_loss: 0.0019 - val_accuracy:
0.9993
Epoch 4/20
640/640 [==============================] - 1s 1ms/step - loss: 0.0043 - accuracy: 0.9987 - val_loss: 0.0065 - val_accuracy:
0.9981
Epoch 5/20
640/640 [==============================] - 1s 2ms/step - loss: 0.0020 - accuracy: 0.9995 - val_loss: 3.5342e-04 - val_accura
cy: 0.9999
Epoch 6/20
640/640 [==============================] - 1s 1ms/step - loss: 0.0042 - accuracy: 0.9989 - val_loss: 7.7324e-04 - val_accura
cy: 0.9999
Epoch 7/20
```

Figure 11: Epoch and batch size

5.3.2 Decision Tree Implementation

This section displays the Decision Tree classifier used and the parameters used. The max_depth for this model is 5 as the tree goes deeper it is less likely for the model from overfitting and the other hyperparameters such as the splitter, ransom_state, presort have taken default values are mentioned in the given figure below.

Decision Tree

```
In [22]: #Decision Tree model
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.metrics import accuracy_score

         #instantiate the model
         tree = DecisionTreeClassifier(max_depth=5)

         #Fitting the model
         tree.fit(x_train, y_train)

Out[22]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
                                max_features=None, max_leaf_nodes=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, presort=False,
                                random_state=None, splitter='best')

In [23]: #predicting the value from the model for the samples
         y_test_tree = tree.predict(x_test)
         y_train_tree = tree.predict(x_train)
```

Figure 12: Decision Tree Implementation

5.3.3 Random Forest Implementation

The below figure shows the implementation of the Random Forest Classifier for this research. In this, the max_depth of the classifier is set to 5 and the steps for the classifier are given in the first and second blocks. The first block comprises the instantiation of the model and the fitting of the model. Once the model is fit using the x_train and the y_train variable set before the prediction can take place using the random forest.

Random Forest Model

In [25]:
```
#Random Forest model
from sklearn.ensemble import RandomForestClassifier

#instantiate the model
forest = RandomForestClassifier(max_depth=5)

#Fitting the model
forest.fit(x_train, y_train)
```

```
C:\Users\Chiku\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: Futu
l change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

Out[25]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=5, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [26]:
```
#predicting the value from the model for the samples
y_test_forest = forest.predict(x_test)
y_train_forest = forest.predict(x_train)
```

Figure 13: Random Forest Implementation

The classifiers and their configurations used are given in the table below:

Classifiers	Configuration	
Random Forest	Function	Parameters
	from sklearn.ensemble import RandomForestClassifier	max_depth=5 random_state=None
Decision Tree	Function	Parameters
	• from sklearn.tree import DecisionTreeClassifier • from sklearn.metrics import accuracy_score	min_samples_split=2 max_depth=5 splitter='best'
Multiple Perceptron	Function	Parameters
	model = tf.keras.Sequential([tf.keras.layers.Dense(hidden_layer_size, input_shape=(input_size,), activation='relu'), # 1st hidden layer tf.keras.layers.Dense(hidden_layer_size, activation='relu'), tf.keras.layers.Dense(hidden_layer_size, activation='relu'), tf.keras.layers.Dense(hidden_layer_size, activation='relu'), tf.keras.layers.Dense(hidden_layer_size, activation='relu'), tf.keras.layers.Dense(hidden_layer_size, activation='relu'), tf.keras.layers.Dense(output_size, activation='softmax') # output layer])	optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'], batch_size = 100, max_epochs = 20

Table 2: Configuration Table

6 Evaluation

The different classifiers used and their accuracy is shown in this section. The main finding can be found in this section where the false positives from the confusion matrix is also given. This is the statistical technique used can help us predict the conclusion of the research.
The output from the confusion matrix can be concluded for the accuracy and performance of the model. These can be calculated from the following formulae to calculate the Accuracy, Precision, and Sensitivity from the confusion matrix (Gad, 2020) (Daniel Gibert, 2019).

$$accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

Figure 14: Accuracy Formula

$$precision = \frac{TP}{TP + FP}$$

Figure 15: Precision Formula

$$Recall = \frac{TP}{TP + FN}$$

Figure 16: Sensitivity Formula

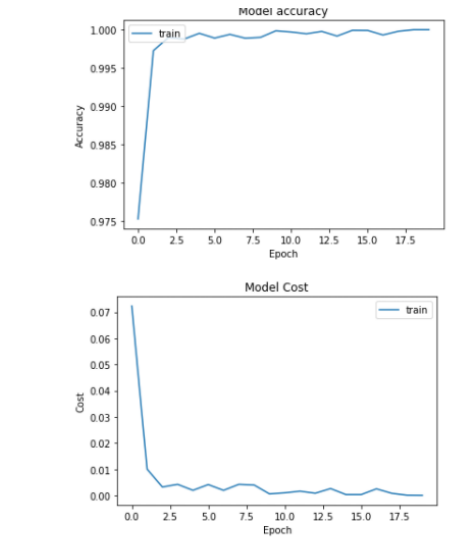
Where TP is true positive, TN is a true negative, FP is False Positive and FN is a false negative. This means that in false positive, benign files are classified as malware and false-negative malware files are classified as benign files. From the confusion matrix, the values for true positives and true negatives are calculated. For this research, accuracy is very important along with precision. Detection of malware with high accuracy and precision will help in the prevention of malware attacks

6.1 Multiple Perceptron Classifier

Using the visualization tools from the matplotlib library and seaborn library, we can visualize the result for the following classifiers. In this, the accuracy and the loss have been calculated and visualized as seen in the bottom figures.

```
In [20]: # Visualize the result
         plt.plot(result.history['accuracy'], label='train')
         plt.legend(loc='upper left')
         plt.title('Model accuracy')
         plt.ylabel('Accuracy')
         plt.xlabel('Epoch')
         plt.show()
         plt.plot(result.history['loss'], label='train')
         plt.legend(loc='upper right')
         plt.title('Model Cost')
         plt.ylabel('Cost')
         plt.xlabel('Epoch')
         plt.show()
```

Figure 17: Code for visualization




```
n [41]: test_loss, test_accuracy = model.evaluate(x_test, y_test)
        print('\nTest loss: {0:.3f}. Test accuracy: {1:.3f}%'.format(test_loss, test_accuracy*100.))

        625/625 [==============================] - 1s 1ms/step - loss: 0.0046 - accuracy: 0.9987: 0s - loss: 0.0046 - accuracy: 0.99

        Test loss: 0.005. Test accuracy: 99.875%
```

Figure 18: model cost and accuracy

We can see that there is a test loss of 0.005 and an accuracy of 99.875%. This accuracy was increased gradually as seen in the graph where we ran 20 epochs to get the specified result. While closely analyzing the result the accuracy grew gradually from 96% to 99.875% which was the result of 20 epochs used as the hyperparameter. This model used minimum resources which are indicated by the test loss and achieved high accuracy in the detection of malware. This is the optimal result that is expected for the detection of malware where the model gave the results with high accuracy and precision.

6.2 Random Forest Classifier

This method fitted several decision trees on the sub-samples of the dataset to average out to improve the predictive accuracy of this classifier and give results. In this, we found an accuracy of 96.1% on the training dataset and an accuracy of 95% on the testing dataset. The results are given in the figure below. This model performed poorly as it did not even cross the benchmark accuracy performed by previous researchers which is 97%. This result is because there might be overfitting in the data which led to the decrease in the accuracy. Using this model will not be efficient enough in the real world as there might be less precision in detecting malware and false alarms will be raised which wastes company resources.

Performance Evaluation

```
In [30]: #Computation of accuracy in Decision tree model

         acc_train_forest = accuracy_score(y_train, y_train_forest)
         acc_test_forest = accuracy_score(y_test, y_test_forest)

         print("Accuracy of Random Forest on the Training dataset: {:.3f}".format(acc_train_forest))
         print("Accuracy of Random Forest on the Teseting dataset {:.3f}".format(acc_test_forest))
```

```
Accuracy of Random Forest on the Training dataset: 0.961
Accuracy of Random Forest on the Teseting dataset 0.950
```

Figure 19: Random Forest accuracy

6.3 Decision Tree Classifier

Similar to the Random Tree classifier, in this, we have found the accuracy of the same. The accuracy in the training dataset and testing dataset came out to be the same which is 98.2%. This model performed a little better than the benchmark accuracy. The MLP model outperformed this model in terms of both accuracy and precision which means that when this model is deployed to detect malware, the MLP model will have more precision and accuracy over the Decision Tree model and will be preferred over this model.

Performance Evaluation

```
In [24]: #Computation of accuracy in Decision tree model

         acc_train_tree = accuracy_score(y_train, y_train_tree)
         acc_test_tree = accuracy_score(y_test, y_test_tree)

         print("Accuracy of Decision tree on the Training dataset: {:.3f}".format(acc_train_tree))
         print("Accuracy of Decision tree on the Teseting dataset {:.3f}".format(acc_test_tree))
```

```
Accuracy of Decision tree on the Training dataset: 0.982
Accuracy of Decision tree on the Teseting dataset 0.982
```

Figure 20: Decision Tree accuracy

6.4 Discussion

In this final step of the evaluation, we compare the machine learning classifiers and check which of them had the highest accuracy. In the below table we can see the results and output of the accuracy of each of the following classifiers.

Classifiers	Training Accuracy	Testing Accuracy
MLP	99.875%	99.875%
Random Forest	96.1%	95%
Decision Tree	98.2%	98.2%

Table 3: Accuracy Measure

From the above result, we can conclude that the best accuracy given by the MLP model is the most accurate from the following classifiers in this case. For this research, we can conclude that the Multiple Perceptron model gave the highest values in accuracy and this also confirms

that the other models tend to overfit the results from the dataset and the values which are outbound. Once this framework is deployed by companies, detection of malware should be done with accuracy and precision which means that there should be no false alarms that could waste precious time and resources. As this is an accuracy-sensitive case, malware cannot have false positives and false negatives as they could be harmful to any system if it affects any. By looking at the data given below by the confusion matrix, we can also calculate the false positives and negatives for this model.

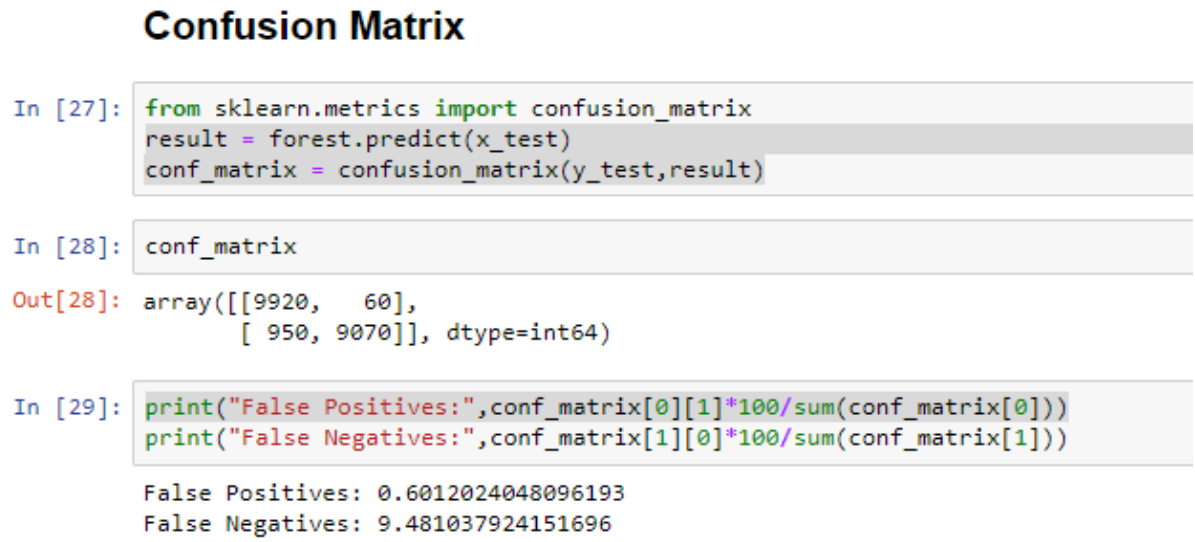


Figure 21: Confusion Matrix

As seen above, the False positive rate is around 0.6% and the false negative has a rate of 9% which means that there is a very low probability of showing even one of these in a real scenario. If we had a bigger dataset this would still have no to some effect on the final output of our scenario. MLP model will detect malware with an accuracy of 99.875% along with a high precision rate of 0.6% of false positives and 9% of false negatives. In other words, once this model is deployed, detection of malware will be done easily with high precision and accuracy and the system will be protected from ransomware.

Method	Dataset (ransomware/benign)	Accuracy
Ban Mohammed Khammas	840/840	97.74%
Proposed Method	50,000/50,000	99.875%

Table 4: Comparison Table with previous research

The above table shows the comparison between this research and the research conducted in a previous research paper. Here the accuracy was improved and this was taken as a baseline for the models in this paper to pass. If the models performed below 97.74%, it was considered as not an efficient model to use for ransomware detection and can be discarded from future research.

7 Conclusion and Future Work

In this research, we were meant to accurately detect Portable Executable files which are none other than malware files using machine learning and have a minimum score of false positives which can give problems in the real world. All of these were achieved in this project and all the classifiers gave potentially good outcomes. The best outcome of accuracy is given by the Multiple Perceptron model with an accuracy of 99.875% and a false positive rate of 0.6%. This suggests that new malware can be detected if using these machine learning algorithms but it would cost a little to the company as these classifiers are heavily dependent upon hardware. Ransomware is a category of malware and is also executed from PE files and this research can proceed toward ransomware with the appropriate dataset focusing on ransomware. Due to time constraints and lack of resources available online, ransomware dataset is not readily available for public use.

The research work can be taken forward to reduce the rate of false negatives which is around 9% in the current build and as it is a nuisance in the real world. Using a bigger dataset with a focus on ransomware can also be done in the future if time permits. For this approach to be commercialized, there need to be added more deterministic mechanisms which will replace the normal anti-viruses with the limitations. In the future, more classifiers can be integrated to compare the margin of accuracy among them.

8 References

Alireza Souri, R. H., 2018. *A state-of-the-art survey of malware detection approaches using data mining techniques,* Iran: Springer Nature.

Asaf Shabtai, U. K. Y. E. C. G. &. Y. W., 2012. *"Andromaly": a behavioral malware detection framework for android devices,* Israel: Springer Nature.

Crane, C., 2021. *thesslstore.* [Online]
Available at: https://www.thesslstore.com/blog/polymorphic-malware-and-metamorphic-malware-what-you-need-to-know/
[Accessed 19 November 2021].

Daniel Gibert, C. M. P., 2019. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications,* 153(1).

Daniele Ucci, L. A. R. B., 2018. Survey on the Usage of Machine Learning Techniques for Malware Analysis. *Computers & Security,* 81(1), pp. 123-147.

Dixon, J., 2021. *crowdstrike.* [Online]
Available at: https://www.crowdstrike.com/cybersecurity-101/ransomware/history-of-ransomware/
[Accessed 22 November 2021].

Fang Zhiyang, J. W. X. K., 2019. Feature Selection for Malware Detection Based on Reinforcement Learning. *IEEE Access,* Volume 7, pp. 176177-176187.

Gad, A. F., 2020. *Evaluating Deep Learning Models: The Confusion Matrix, Accuracy, Precision, and Recall.* [Online]

Available at: https://blog.paperspace.com/deep-learning-metrics-precision-recall-accuracy/
[Accessed 6 December 2021].
Harford, I., 2021. *techtarget.* [Online]
Available at: https://www.techtarget.com/searchsecurity/feature/The-history-and-evolution-of-ransomware
[Accessed 21 November 2021].
IBM, 2021. *CRISP-DM Help Overview.* [Online]
Available at: https://www.ibm.com/docs/en/spss-modeler/SaaS?topic=dm-crisp-help-overview
[Accessed 12 December 2021].
Intel, 2012. *Intel.* [Online]
Available at: https://www.intel.com/content/www/us/en/products/systems-devices/sku/h92638939/lenovo-legion-y540/options.html
[Accessed 6 December 2021].
Kaggle, 2021. *Kaggle.* [Online]
Available at: https://www.kaggle.com/getting-started/104154
[Accessed 6 December 2021].
Lenovo, 2019. *Lenovo.* [Online]
Available at: https://www.lenovo.com/ie/en/laptops/legion-laptops/legion-y-series/Lenovo-Legion-Y540-15/p/88GMY501214
[Accessed 6 December 2021].
Numpy, 2010. *Numpy ORG.* [Online]
Available at: https://numpy.org/doc/stable/user/absolute_beginners.html
[Accessed 6 December 2021].
Sherif Saad, W. B. H. E., 2019. The Curious Case of Machine Learning In Malware Detection. *5th International Conference on Information Systems Security and Privacy, 2019,* p. 9.
Shetty, S., 2018. *Why TensorFlow always tops machine learning and artificial intelligence tool surveys.* [Online]
Available at: https://hub.packtpub.com/tensorflow-always-tops-machine-learning-artificial-intelligence-tool-surveys/
[Accessed 7 December 2021].
Y. Ye, L. W. K. H., 2018. A Risk Classification Based Approach for Android Malware Detection. *KSII Transactions on Internet and Information Systems,* Volume 11, pp. 959-981.
Ye Wu, R. R., 2012. *7 Techniques to Handle Imbalanced Data.* [Online]
Available at: https://www.kdnuggets.com/2017/06/7-techniques-handle-imbalanced-data.html
[Accessed 6 December 2021].
Z. Bazrafshan, H. H. S. M. H. F. a. A. H., 2016. A Comparison of Malware Detection Techniques Based on Hidden Markov Model. *Journal of Information Security,* 7(3), pp. 215-223.